NC ASSIGNMENT # 01

Name : Muhammad Umair
SId : 7791

Question :

Write the code of following root finding methods. Display result of all iterations in Pandas DataFrame and plot graph to represent value of root and function for assigned interval.

1. Bisection Method
2. Regula Falsi
3. Secant
4. Newton Raphson
5. Muller Plot a bar graph that will compare no. of iterations for all above methods.

Important Libraries :

In [11]:
```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
from scipy.misc import derivative
```

Function :

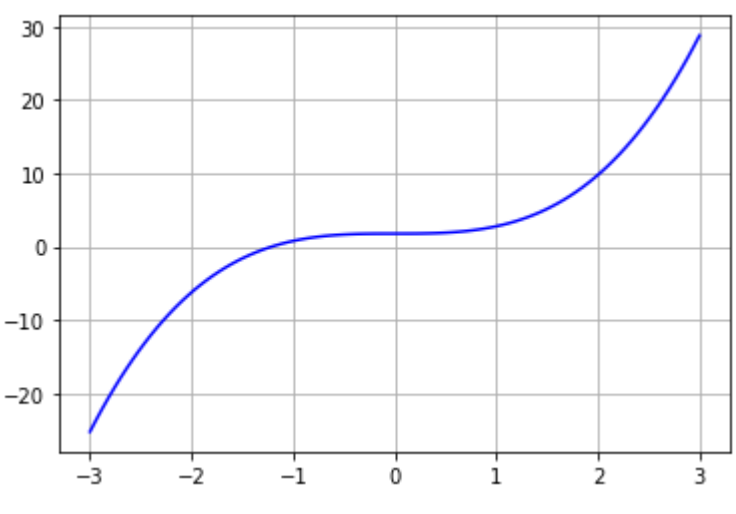
In [12]:
```
def f(x):

        return x**3 + 1.761
```

Graph Of Function :

In [13]:
```
x = np.linspace(-3,3,100)
fig = plt.figure()
plt.plot(x,f(x), 'b')
plt.grid()


plt.show()
```



In

Bisection Method :

Out[15]:

In [14]:

```
alst=list()
blst=list()
midlst=list()
ilst=list()
def Bisection(a,b):
    i=0
    while(np.abs(a-b))>0.001:
        mid = (a+b)/2


        if(f(a)*f(mid))<0:
            b=mid

        else:
            a=mid

        i+=1
        ilst.append(i)
        blst.append(b)
        alst.append(a)
        midlst.append(mid)


    return mid
```

In [15]:

```
root=Bisection(-2,2)
BisecIteration=len(ilst)

BisectionData={
    'ilst':ilst,
    'A':alst,
    'B':blst,
    'Mids':midlst


}
BisectionData=pd.DataFrame(BisectionData)
BisectionData.transpose()
```

	0	1	2	3	4	5	6	7	8	9	10	11
ilst	1.0	2.0	3.0	4.00	5.000	6.0000	7.00000	8.000000	9.000000	10.000000	11.000000	12.000000
A	-2.0	-2.0	-1.5	-1.25	-1.250	-1.2500	-1.21875	-1.218750	-1.210938	-1.210938	-1.208984	-1.208008
B	0.0	-1.0	-1.0	-1.00	-1.125	-1.1875	-1.18750	-1.203125	-1.203125	-1.207031	-1.207031	-1.207031
Mids	0.0	-1.0	-1.5	-1.25	-1.125	-1.1875	-1.21875	-1.203125	-1.210938	-1.207031	-1.208984	-1.208008

In

Regula Falsi :

Out[16]:

[16]:

```
1
2  alst=list()
3  blst=list()
4  rootlst=list()
5  ilst=list()
6
7  def RegularFalse(a,b):
8      i=0
9
10     while(i<200):
11         Xr=(a*f(b)-b*f(a))/(f(b)-f(a))
12         if(Xr == 0 or np.abs(f(Xr)) < 0.001):
13             break
14         if(f(a) *f(Xr))<0:
15             b=Xr
16         else:
17             a=Xr
18         i+=1
19
20
21         alst.append(a)
22         blst.append(b)
23         rootlst.append(Xr)
24         ilst.append(i)
25
26
27
28     return i,Xr
29 RegularFalse(-2,2)
30 Rftottal=len(ilst)
31
32 rfData={
33     'No Of Iterations':ilst,
34     'A lst':alst,
35     'B lst':blst,
36     'Root lst':rootlst
37
38 }
39 RegularFalsi=pd.DataFrame(rfData)
40 RegularFalsi.transpose()
```

	0	1	2	3	4	5	6	7	8	9	10
No Of Iterations	1.00000	2.000000	3.000000	4.000000	5.000000	6.000000	7.000000	8.000000	9.000000	10.000000	11.000000
A lst	-2.00000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000
B lst	-0.44025	-0.770476	-0.982979	-1.099998	-1.158029	-1.185199	-1.197566	-1.203121	-1.205602	-1.206706	-1.207198
Root lst	-0.44025	-0.770476	-0.982979	-1.099998	-1.158029	-1.185199	-1.197566	-1.203121	-1.205602	-1.206706	-1.207198

In

Secant:

In

```
ilst=list()
plst=list()


def secant(p0, p1):
    for i in range(0,100):

        p = p1 - ((f(p1)*(p1-p0))/(f(p1) - f(p0)))




        ilst.append(i+1)
        plst.append(p)

        if abs(p-p1) < 0.001:

            return p
        p0 = p1
        p1 = p
    return "Them method failed after {} iterations">format(N)
secantData={
    'ilst':ilst,
    'plst':plst

}
secant(-2,2)
secanttottal=len(ilst)
secant=pd.DataFrame(secantData)
secant.transpose()
```

Out[17]:

	0	1	2	3	4	5	6	7
ilst	1.00000	2.000000	3.000000	4.000000	5.000000	6.000000	7.000000	8.000000
plst	-0.44025	-0.945988	-1.553517	-1.137377	-1.190293	-1.208647	-1.207576	-1.207591

Newton Raphson :

In [18]:

```
x=-2
xlst=list()
ilst=list()
for i in range(0,100):
    if(abs(f(x))>0.001):

        x = x- (f(x) / derivative(f , x))
        xlst.append(x)
        ilst.append(i+1)

    else:
        xlst.append(x)
        ilst.append(i+1)
        break
NewtonData={
    'No Of Iterations':ilst,
    'Roots':xlst
}
NewtonData=pd.DataFrame(NewtonData)

NewtonTottal=len(ilst)
NewtonData.transpose()
```

Out[18]:

	0	1	2	3	4	5	6
No Of Iterations	1.000000	2.00000	3.000000	4.000000	5.000000	6.00000	7.00000
Roots	-1.520077	-1.29928	-1.227986	-1.211559	-1.208336	-1.20773	-1.20773

In

Muller Method :

In [25]:

```
plst=list()
ilst=list()
def MullerM(pa,pb,pc):
    p0 = pa
    p1 = pb
    p2 = pc
    f0 = f(p0)
    f1 = f(p1)
    f2 = f(p2)
    i = 0
    while i<=100:
        c = f2
        b = ((p0-p2)**2 *(f1-f2)-(p1-p2)**2 *(f0-f2))/((p0-p2)*(p1-p2)*(p0-p1))
        a = ((p1-p1)*(f0-f2) - (p0-p2)*(f1-f2))/((p0-p2)*(p1-p2)*(p0-p1))
        p3 = p2 - 2*c/(b+(b/abs(b))*np.sqrt(b**2 -4*a*c))
        plst.append(p3)
        ilst.append(i+1)
        if abs(p3-p2)<0.01:
            return p3
        p0 = p1
        p1 = p2
        p2 = p3
        f0 = f(p0)
        f1 = f(p1)
        f2 = f(p2)
        i = i+1

MullurData={
    'No Of Iterations':ilst,
    'Roots':plst

}

```

In [26]:

```
MullerM(-3,3,1)
MullurData=pd.DataFrame(MullurData)

MullerTottal=len(ilst)
MullurData.transpose()
```

Out[26]:

	0	1	2	3	4	5	6	7
No Of Iterations	1.000000	2.000000	3.000000	4.000000	5.000000	6.000000	7.000000	8.000000
Roots	0.735187	-0.264441	0.505372	-1.023458	-1.476976	-1.164584	-1.204279	-1.207626

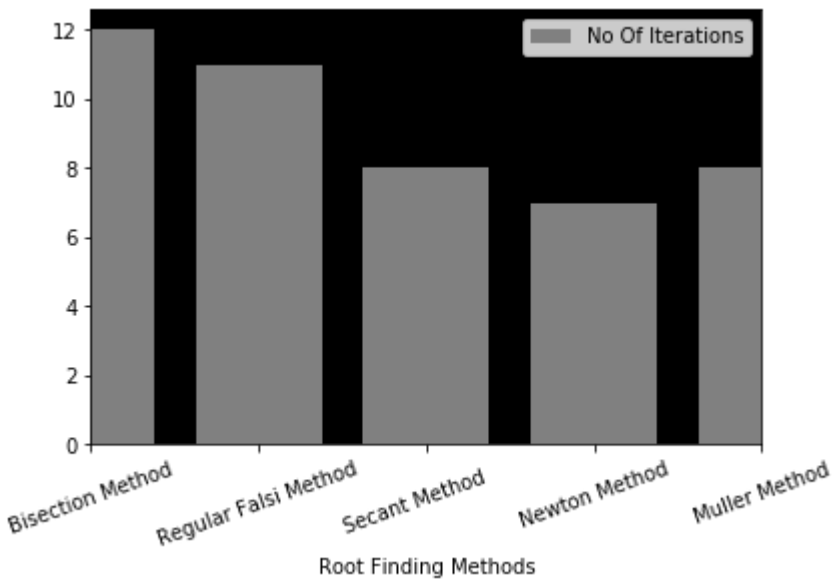
In

Bar Graph :

In [27]:

```
df = pd.DataFrame({'Root Finding Methods':['Bisection Method', 'Regular Falsi Method','Secant Method    ','N
ax = df.plot.bar(x='Root Finding Methods', y='No Of Iterations',rot=380, align='center', color=('gray'),widt
ax.set_facecolor("black")


```



Bar Graph Horizontally :

[28]:

```
Methods = ['Bisection Method','Regular Falsi Method','Secant Method    ','Newton Method','Muller Method']
NoOfIterations = [BisecIteration, Rftottal,secanttottal,NewtonTottal,MullerTottal]


plt.barh(Methods, NoOfIterations, color='gray')
plt.xlabel("No Of Iteration")
plt.ylabel("Methods")

plt.show()
```